Support Vector Machine w/ Fisher Linear Discrminant vs. Fast Gradient Sign Attack

Armando Herrera
Computer Science
The University of Texas Rio Grande Valley
Edinburg, Texas
armando.herrera02@utrgv.edu

***Abstract*— Here, we use a Support Vector Machine (SVM) with and without Fisher Discriminant Analysis pruning to create a classification model. A Fast Gradient Sign Attack (FGSM) is then done. While the classification model first obtained 95% accuracy, the attack rendered the classifier useless with an accuracy of 10%. Various settings with an FDA pruner are also tested.**

Keywords—Support Vector Machines, Fisher Linear Discriminant Analysis, Fast Gradient Sign Attack, Stochastic Gradient Descent,

I. Introduction

A. Support Vector Machines

An SVM is a supervised learning model which analyzes data for classification and regression. In the field of machine learning, we know that SVMs are a robust method of classification and regression obtaining accuracy rival only to that of Deep Neural Networks. An SVM, in a classifier, is a binary classifier, so it classifies between one category and another. In a more mathematical description, an SVM creates a hyperplane, a plane in n-dimensional space. This hyperplane separates the object's class. An SVM can either have a hard or soft-margin. With a hard-margin, when training, an object with a certain class isn't allowed be within the hyperplane's margins while, in a soft margin, the margins are more flexible. Sometimes, when a dataset isn't linearly separable, the SVM will perform poorly. To fix this a kernel is used, it increases the dimensionality of the hyperplane, allowing for the separation of the classes; this is called the kernel trick. There are various methods for finding this hyperplane that separates all classes. One way described in the SVM Wikipedia page uses the hinge loss and reduces the SVM to a quadratic programming problem. More modern methods exist such as, Sub-gradient descent where Stochastic Gradient Descent (SGD) or Gradient Descent (GD) are adapted with the quadratic programming problem, in this paper, I do something like this.

In terms of the Kernel Trick, there are various methods to perform this function, In an SVM, the inner product is computed between a weights vector and an input vector with an increase of dimensionality then a bias is added. The kernel function increases the dimensions in which the weights and the input vector operate by replacing the dot product with a function.

$$k(x, x') = < \varphi(x), \varphi(x') >$$

The kernel most in use, for a good reason, is called the Radial Basis Function (RBF) kernel.

$$k(x, x') = \exp(-\gamma||x - x'||^2)$$

Where γ is a tunable floating-point hyper-parameter between 0 and 1. For when there are many samples and in cases where there is a large dimensionality there exists approximations of the kernel functions where a function $z(x)$ is created.

$$k(x, x') = < \varphi(x), \varphi(x') > \approx < z(x), z(x') >$$

For the RBF kernel there are two main functions that can be used in place of the dot product, one is called the Nyström method and the other is referred to as the Kitchen Sink Method and uses a Monte Carlo method along with Fourier transformations. Since I use the Kitchen Sink Method in this project, I will go into more details in the model section.

To account for multiple classes there, exist three main methods of applying and having an SVM classify multiple classes. One way, and the method used in this project, is called "one vs. all" method where an SVM is created for each class and each class is pitted against all other classes. In other words, each SVM classifies whether it is that class or it is not that class. The max argument is then taken to be the most likely class. The second way "one vs. one" where an SVM is created for each possible combination of classes and an argmax of an aggregated score is used to classify. The third method is called a Directed Acyclic Graph which involves a tree structure of SVMs.

B. Why matrix multiplication makes for an SVM

One of the reasons that a "one vs. all" scheme is used to create the SVMs is because of our ability to generalize and have all SVM's dot, or kernel, operations done in one fell, matrix multiplication, operation.

proof. A single SVM's hyperplane is mathematically defined below.

$$\vec{w} \cdot \vec{x} - b = 0$$

Where $\vec{w}$ is the weights vector, $\vec{x}$ is the input vector, and b is a bias scalar. Let's say that there are 3 classes we want to differentiate from, and we go with a "one vs. all" approach we would then create three different SVMs.

$$\overrightarrow{w_1} \cdot \overrightarrow{x_1} - b_1 = 0$$
$$\overrightarrow{w_2} \cdot \overrightarrow{x_2} - b_2 = 0$$
$$\overrightarrow{w_2} \cdot \overrightarrow{x_2} - b_2 = 0$$

We can then reduce the dot product operations to its base components.

$$\sum_{i=1}^{n} w_{1i}x_{1i} - b_1 = 0$$
$$\sum_{i=1}^{n} w_{2i}x_{2i} - b_2 = 0$$
$$\sum_{i=1}^{n} w_{3i}x_{3i} - b_3 = 0$$

Matrix multiplication is defined as

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

Notice, since there is only one value for the index in the summations for the SVM that are not part of the summation, we can switch around the indexes to mimic the Matrix Multiplication by changing the order of operations.

$$\sum_{i=1}^{n} w_{1i}x_{i1} - b_1 = 0$$
$$\sum_{i=1}^{n} w_{2i}x_{i2} - b_2 = 0$$
$$\sum_{i=1}^{n} w_{3i}x_{i3} - b_3 = 0$$

Notice again, since the enumeration of the SVM are increasing from 1 to 3 we can then represent the series of SVMs in terms of Matrix Multiplication.

$$\sum_{i=1}^{n} w_{ji}x_{ij} - b_j = 0 \mid j \in [1,3] \cap Z^+$$

$\square$

C. *Fisher Linear Discriminant Analysis (FDA)*

FDA is a method used to find a linear combination of features that separate two classes. This is used as a linear classifier and for dimensionality reduction before further classification. FDA starts with an object function.

$$J(w) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{S}_1^2 + \tilde{S}_2^2}$$

Where S_1 and S_2 are defined as the scatter matrices of each class. The objective function is then used to solve for a, v, line that best separates the classes by solving a generalized eigenvalue problem.

$$S_B v = \lambda S_W v$$

The solution is shown below.

$$v = S_W^{-1}(\mu_1 - \mu_1)$$

This method has the same problem as the SVM, the FDA is a binary classifier not a multi-class classifier. There is a method to generalize an FDA which is called Multiple Discriminant Analysis but, in this project, I stuck with using many FDAs with the "one vs all" scheme.

D. *Stochastic Gradient Descent (SGD)*

SGD is an iterative optimization algorithm. SGD is derived from Gradient Descent (GD). Unlike GD, SGD instead of calculating the gradient explicitly, calculates an estimate of the gradient.

$$w \coloneqq w - \eta \nabla Q_i(w)$$

The approximation for the gradient is, typically, calculated, not with numerical differentiation, but with back-propagation. Back-propagation is a, quick, method of calculating both the value and a gradient in one fell swoop using the chain rule from Calculus; Automatic Differentiation is a generalization of back-propagation.

E. *Fast Gradient Sign Attack (FGSM)*

The FGSM attack is described in a paper title, "Explaining and Harnessing Adversarial Examples". The paper shows that by calculating the gradient, $\nabla_x J(\theta, x, y)$, through a model, neural network or SVM, of an image, through backpropagation, and by adding the sign of the gradient multiplied by a hyper-parameter, ϵ, we can decrease the accuracy of the models that rely heavily on linear elements.

$$f(x) = x + \epsilon sign(\nabla_x J(\theta, x, y))$$

F. *PyTorch*

PyTorch is an open source machine learning library for Python based on Torch. It is used in many applications such as Computer Vision. It was developed and is maintained by Facebook and a vibrant open-source community around it. It is well known for being dynamic in that you can perform any differentiable operations and you can use PyTorch to easily backpropagate and calculate gradients using something known as a Gradient Tape. A Gradient Tape is a sort of tape that records all operations performed on a designated tensor, matrix, vector, or scalar. This make it one of a kind and this Gradient Tape is yet to be implemented in many places, including MATLAB. TensorFlow is the only other implementation of the Gradient Tape that I know. They implemented it in their 2.0 release this September. This Gradient Tape tool is a strong tool for researchers!

II. Model

A. *FDA*

Here, an FDA based model is used to make a rough calculation of linear separation between classes in a "one vs all" scheme. With this in mind, we can make some optimization to the calculations via memorization. The scatter matrix for each class is first calculated.

$$S_i = \sum_{x_j \in Class\ i} (x_{ij} - \mu_i)(x_{ij} - \mu_i)$$

Where x_{ij} is the sample from the ith class and the jth sample in that class and μ_i is the mean of that class. Then S_w, the within class scatter matrix is calculated by summing all the scatter matrices.

$$S_w = \sum_{i=1}^{c} S_i$$

Where c is the number of classes. Then, using the inverse of S_w the discriminant line is calculated on a per class basis.

$$v_i = S_w^{-1}(\mu_i - \frac{1}{c-1}(\sum_{j=0}^{i-1}\mu_j + \sum_{j=i+1}^{c}\mu_j))$$

Then to calculate the distance from the line on a per class basis.

$$y_i = v^T c_i^T$$

The closest n percentage of points is then used in the training of the SVM.

B. SVM

The SVM model used is the one described in the proof in the SVM section of the Introduction where a weight matrix, $w \in R^{m\times n}$, and a bias vector, $b \in R^m$, is defined. Let m be the number of classes and n be the number of dimensions the kernel outputs. We then, according to our kernel approximation, define random weights matrix offsets, $w_{offset} \in R^{m\times n} \cap [-\sqrt{2\gamma}, \sqrt{2\gamma}]$, and a random bias vector offsets, $b_{offset} \in R^m \cap [0,2\pi]$, of the same size. The weight was randomly initiated with real number in the interval $[-\sqrt{2\gamma}, \sqrt{2\gamma}]$ and the bias in the interval $[0,2\pi]$. The kernel function is as follows.

$$z(x) = \frac{\sqrt{2}}{N}\cos(xw_r + b_r)$$

Where N is the number of output kernel dimensions and x is the input sample. The SVM output operation is, then, shown below.

$$f(x) = wz(x) + b$$

To train the SVM, SGD was the optimization algorithm of choice, Since the output of the kernel function is not differentiable natively the cross-entropy loss function was used. It is defined below.

$$H(p,q) = -\sum_i p_i log q_i$$

Where H is the cross-entropy, p is a binary indicator and p is the predicated probability.

More details on the model and implementation can be obtained via the GitHub project.

III. Methods

While testing we noticed that tuning hyper-parameters was an incredibly long and tedious task, so Optuna, an advanced hyper-parameter optimization library, was used to tune hyper-parameters not otherwise mentioned. On the first run all hyper-parameters were tuned as to find the best possible accuracy the SVM could have gotten with all data. Subsequently, only the learning rate, the gamma, and the number of kernel dimensions were tuned using the already optimal number of epochs. A set of benchmarks were then done with varying cutoff percentages. These benchmarks started with 25 trials of hyper-parameter optimizations using Optuna, followed by a timed training with the found optimal parameters. Testing accuracy and training time was measured and recorded. Since the cutoff rates varied the number of samples used for training consequently varied, this was recorded as well.

The benchmark starts with a cutoff rate of 0, so all samples are used to train the SVM followed by an optimization, training, testing, and recording at every 0.2 intervals until reaching 0.8. The data tested was the MNIST handwritten digits dataset from the National Institute of Standards and Technology. At each step the model was then attack via FGSM and the resulting accuracy is recorded.

IV. Results

TABLE I. RESULTS FROM BENCHMARKS

Accuracy	FDA Cut off	Sample Count	Attacked Accuracy	Training Time (s)
95.02	0.0	60000	12.93	204.8988
91.03	0.2	35044	15.33	114.1261
74.28	0.4	15331	20.09	51.98336
54.43	0.6	4609	22.64	15.89015
34.71	0.8	538	9.040	2.103744

In the table above, as the FDA cut off increased by 0.2 the number of samples decreased by half. The second benchmark with a cut from 0.2 was able to reduce the number of samples by half which resulting in half the training time while keeping the accuracy over 90%. On the third benchmark the accuracy decreased to 74.28% while reducing the number of samples used to train and the training time of the SVM by 1/3. All other benchmarks resulted in an SVM below 70% accuracy.

Oddly, As the FDA Cut off increased the attack's edge was blunted slightly. For the first 4 benchmarks, as we increased the cut off the accuracy after attack increased from 12.93% to 22.64%. This is interesting, and I'll conjecture as to what could possibly cause this in the conclusion.

V. Conclusion

In conclusion, an SVM was created and trained using data pruned with an FDA, this decreased training time drastically while only decreasing accuracy slightly. There is a point where pruning more samples decreases the accuracy drastically. In addition, the accuracy of perturbed images increases as data is pruned until a threshold is reached, after which, the accuracy decreases again substantially.

One of the possible reasons that the accuracy of the perturbed images increases as the number of images pruned with FDA may have something to do with the boundary between classes in hyper-dimensional space. The use of points, filtered with FDA, known to be a good representation of said class could blunt the attack. The reduction of overfitting within the model could also be a factor in the mitigation of the attack and a model with more generalization capabilities is more resistant to the attacks and is a good future research avenue.

References

[1] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic diffentiation in PyTorch," *Conference on Neural Information Processing Systems*, Oct. 2017.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversaial Examples." 20-Mar-2015.

[3] J. Xu, Y. Lu, and B. Zou, "Learning performance of DAGSVM algorithm based on Markov sampling," *2015 International Conference on Machine Learning and Cybernetics (ICMLC)*, 2015.

[4] M. Welling, "Fisher Linear Discriminant Analysis." University of California, Irvine, 2009.

[5] S. Patel, "Chapter 2 : SVM (Support Vector Machine) - Theory," *Medium*, 04-May-2017. [Online]. Available: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.

[6] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter, "Pegasos: primal estimated sub-gradient solver for SVM," *Mathematical Programming*, vol. 127, no. 1, pp. 3–30, 2010.

[7] "Stochastic gradient descent," *Wikipedia*, 08-Dec-2019. [Online]. Available: https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[8] "Support-vector machine," *Wikipedia*, 01-Dec-2019. [Online]. Available: https://en.wikipedia.org/wiki/Support-vector_machine.

[9] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna," Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '19, 2019.

[10] T. Hofmann, B. Scholkopf, and A. J. Smola, "Kernel Methods in Machine Learning," *The Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, May 2008.

[11] Y. LeCun, C. Cortes, and C. J. C. Burges, "MNIST handwritten digit database," 2010.